EAP-TTLS の ProVerif による評価結果

国立研究開発法人 情報通信研究機構

1. 基本情報

- ✧ 名前

  Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)

- ✧ 機能

  EAP において、TLS を用いて使用する暗号アルゴリズムのネゴシエーションとサーバ認証・鍵交換を実現し、ユーザ名とパスワードを用いてクライアント認証を実現すること目的としたプロトコル。

- ✧ 関連する標準

  RFC5281 (https://tools.ietf.org/html/rfc5281)

2. ProVerif の文法による記述

CPVP 技術文書「**EAP-TTLS の概要**」で示したロール S(Access Point)、ロール T(TTLS Server)、ロール A(AAA/H)を単一のプロセスとして扱い、ロール A (Authenticator) で表す。

2.1. プロトコル仕様

```
(*
AVISPA version of EAP-TTLS [RFC5281]
*)

set selFun = Nounifset.
set stopTerm = false.

(*
set traceDisplay = long.
*)
```

```
free c: channel.

type skey.
type pkey.
type symkey.
type nonce.
type host.

(* consts*)
const request_id: bitstring [data].
const response_id: bitstring [data].
const start_ttls: bitstring [data].
const success: bitstring [data].
const Shd: bitstring [data].
const Ccs: bitstring [data].
const ttls_challenge: bitstring [data].
const master_secret: bitstring [data].
const key_expansion: bitstring [data].
const ttls_keying_material: bitstring [data].

(* PKE *)
fun pk(skey): pkey.
fun encrypt(bitstring, pkey): bitstring.
reduc forall x: bitstring, sk: skey;
      decrypt(encrypt(x, pk(sk)), sk) = x.

(* SKE *)
fun sencrypt(bitstring, symkey): bitstring.
reduc forall x: bitstring, k: symkey;
      sdecrypt(sencrypt(x, k), k) = x.

(* Sig *)
```

```
fun sign(bitstring, skey): bitstring.
reduc forall x: bitstring, sk: skey;
      check(sign(x, sk), pk(sk)) = x.
reduc forall x: bitstring, sk: skey;
      getmsg(sign(x, sk)) = x.

(* functions for describing secrecy *)
(*
fun b2symkey(bitstring): symkey.
reduc forall x: bitstring, k: bitstring;
      hide(x, k) = sencrypt(x, b2symkey(k)).
*)

(* PRF and Hash *)
fun PRF(bitstring): symkey.
(*
fun KeyGen(bitstring): symkey.
*)
reduc forall h: host, np: nonce, ns: nonce, ms: symkey;
      KeyGen(h, np, ns, ms) = PRF((h, ms, key_expansion, np, ns)).
fun H(symkey): bitstring.
fun CHAP_PRF_challenge(bitstring): bitstring.
fun CHAP_PRF_id(bitstring): bitstring.
fun CHAP_algorithm(host, bitstring, bitstring): bitstring.

(* tables *)
table keys(host, pkey).
table users(host, bitstring).

(* events *)
event beginS(host, host, symkey).
event beginP(host, host, symkey).
```

```
event endS(host, host, symkey).
event endP(host, host, symkey).
event end().

(* free names *)
free hostS, hostP, anonymous: host.
free userP, userQ: host.
free passwdP, passwdQ: bitstring [private].

weaksecret passwdP.
(*
weaksecret passwdQ.
*)

(* query *)
free secretPMSK, secretSMSK, secretTest: bitstring [private].

query attacker(secretPMSK).
query attacker(secretSMSK).
(*
query attacker(secretTest).
*)

query p: host, s: host, msk: symkey;
      inj-event(endS(p, s, msk)) ==> inj-event(beginP(p, s, msk)).
query p: host, s: host, msk: symkey;
      inj-event(endP(p, s, msk)) ==> inj-event(beginS(p, s, msk)).


let procP(UserId: host, UserName: host, Password: bitstring, pkCA: pkey)
=
    in(c, (Version: bitstring, CipherSuite: bitstring, SessionID:
```

```
bitstring));
    let P = UserId in
    (* EAP Start *)
    in(c, =request_id);
    out(c, (response_id, UserId));
    (* 1st phase: TLS *)
    in(c, =start_ttls);
    new Np: nonce;
    out(c, (Version, SessionID, Np, CipherSuite));
    in(c, (=Version, =SessionID, Ns: nonce, =CipherSuite,
           certS: bitstring, Ske: bitstring, =Shd));
    let (S: host, pkS: pkey) = check(certS, pkCA) in
    new PMS: bitstring;
    let MS = PRF((PMS, master_secret, Np, Ns)) in
    let ClientK = KeyGen(P, Np, Ns, MS) in
    (*通信相手 S が特定のサーバ hostS であることをユーザが確認する場合はここで S=hostS をチェックする ★ *)
    if S = hostS then
    out(c, (encrypt(PMS, pkS), Ccs, sencrypt(H(MS), ClientK)));
    in(c, (=Ccs, encHMS_s: bitstring));
    let ServerK = KeyGen(S, Np, Ns, MS) in
    if H(MS) = sdecrypt(encHMS_s, ServerK) then
    (* 2nd phase: using tunneling to authenticate peer*)
    let CHAP_challenge = CHAP_PRF_challenge((MS, ttls_challenge, Np, Ns)) in
    let ChapId = CHAP_PRF_id((MS, ttls_challenge, Np, Ns)) in
    let ChapRs = CHAP_algorithm(UserName, Password, CHAP_challenge) in
    let MSK = PRF((MS, ttls_keying_material, Np, Ns)) in (* MSK can be generated now*)
    event beginP(UserName, S, MSK);
    out(c, sencrypt((UserName, CHAP_challenge, (ChapId, ChapRs)), ClientK));
```

```
    in(c, =success);
    (* for describing security *)
    (*通信相手 S が特定のサーバ hostS であることをユーザが確認しない場合はここで S=hostS をチェックする ★ *)
    (*
    if S = hostS then
    *)
    out(c, sencrypt(secretPMSK, MSK));
    event endP(UserName, S, MSK);
    event end().

let procS(S: host, skS: skey, certS: bitstring) =
    in(c, (Version: bitstring, CipherSuite: bitstring, SessionID: bitstring));
    (* EAP Start *)
    out(c, request_id);
    in(c, (=response_id, P: host));
    out(c, start_ttls);
    (* 1st phase: TLS *)
    in(c, (=Version, =SessionID, Np: nonce, =CipherSuite));
    new Ns: nonce;
    new Ske: nonce;
    out(c, (Version, SessionID, Ns, CipherSuite,
            certS, Ske, Shd));
    in(c, (encPMS: bitstring, =Ccs, encHMS_c: bitstring));
    let PMS = decrypt(encPMS, skS) in
    let MS = PRF((PMS, master_secret, Np, Ns)) in
    let ClientK = KeyGen(P, Np, Ns, MS) in
    if H(MS) = sdecrypt(encHMS_c, ClientK) then
    let ServerK = KeyGen(S, Np, Ns, MS) in
    out(c, (Ccs, sencrypt(H(MS), ServerK)));
    (* 2nd phase: using tunneling to authenticate peer*)
```

```
    in(c, auth: bitstring);
    let (UserName: host, CHAP_challenge: bitstring, (ChapId: bitstring,
ChapRs: bitstring))
        = sdecrypt(auth, ClientK) in
    get users(=UserName, Password) in
    if CHAP_challenge = CHAP_PRF_challenge((MS, ttls_challenge, Np,
Ns)) then
    if ChapId = CHAP_PRF_id((MS, ttls_challenge, Np, Ns)) then
    if ChapRs = CHAP_algorithm(UserName, Password, CHAP_challenge) then
    out(c, success);
    (* for describing security *)
    let MSK = PRF((MS, ttls_keying_material, Np, Ns)) in (* MSK can be
generated now*)
    event beginS(UserName, S, MSK);
    if UserName = userP then
    event endS(UserName, S, MSK);
    out(c, sencrypt(secretSMSK, MSK));
    event end().

let userRegistration =
    in(c, (UserName: host, Password: bitstring));
    if UserName <> userP then
    insert users(UserName, Password).



let procCA(skCA: skey) =
    in(c, (h: host, k: pkey));
    if h <> hostS then
    out(c, sign((h, k), skCA)).

process
        (* CA *)
```

```
        new skCA: skey;
        let pkCA = pk(skCA) in
        out(c, pkCA);
        (* Server *)
        new skS: skey;
        let pkS = pk(skS) in
        let certS = sign((hostS, pkS), skCA) in
        insert keys(hostS, pkS);
        out(c, certS);
        (* Users *)
        insert users(userP, passwdP);
        insert users(userQ, passwdQ);
        (
                (!procP(anonymous, userP, passwdP, pkCA))|  (* using
anonymous. see Sect7.2 in RFC  *)
                (!procS(hostS, skS, certS))|
                (!procCA(skCA))|
                (!userRegistration)
        )
```

2.2. 攻撃者モデル

上述の記述に含まれる。

2.3. セキュリティ要件

上述の記述の (* queries *) に該当する。

```
(* (1) *)
query p: host, s: host, msk: symkey;
      inj-event(endS(p, s, msk)) ==> inj-event(beginP(p, s, msk)).

(* (2) *)
query p: host, s: host, msk: symkey;
      inj-event(endP(p, s, msk)) ==> inj-event(beginS(p, s, msk)).
```

```
(* (3) *)
query attacker(secretPMSK);
      attacker(secretSMSK).

(* (4) *)
weaksecret passwdP.
```

(1) ロール P(Peer)がロール A(Authenticator)を正しく認証すること。
すなわち、ロール P(Peer)が実行を完了したとき、同じパラメータを用いて実行を完了したロール A(Authenticator)が存在する。

(2) ロール A(Authenticator)がロール P(Peer)を正しく認証すること。
すなわち、ロール A(Authenticator)が実行を完了したとき、同じパラメータを用いて実行を完了したロール P(Peer)が存在する。

(3) 交換したセッション鍵(MS)の秘匿性。

(4) CHAP のパスワード(passwdP)へのオフライン推測攻撃への耐性。

3. ProVerif による評価結果

3.1. 出力

安全性はいずれも成り立つ（true）という結果が出力された。

```
RESULT inj-event(endP(p,s,msk)) ==> inj-event(beginS(p,s,msk)) is false.
RESULT (even event(endP(p_2930,s_2931,msk_2932)) ==> event(beginS(p_2930,s_2931,msk_2932)) is false.)
RESULT inj-event(endS(p_3506,s_3507,msk_3508)) ==> inj-event(beginP(p_3506,s_3507,msk_3508)) is true.
RESULT not attacker(secretSMSK[]) is true.
RESULT not attacker(secretPMSK[]) is true.
RESULT Weak secret passwdP is true (bad not derivable).
```

3.2. 攻撃の解説

前述のように、攻撃は発見されなかった。

4. 形式化

4.1. 方針

AVISPA ライブラリと同様にモデル化した。公開鍵暗号、秘密鍵暗号、電子署名、疑似乱数、ハッシュ関数のみを含むため、Dolev-Yao モデルで記述できた。

4.2. 妥当性

特になし。

4.3. 検証ツールとの相性

プロトコル仕様、攻撃者モデル、セキュリティ要件を ProVerif で記述するにあたって、特に制限はなかった。ただし、デフォルトの評価では評価が完了しなかったため、参加者が型チェックを行うと仮定して(set ignoreTypes=false) 評価した。

4.4. 検証ツール適用時の性能

検証時間は 0.1 秒未満であった。実行環境は以下のとおり。

- ✧ Intel Core i7 L620 2.00HGz
- ✧ Windows7 上の VirtualBox 仮想マシン上の Ubuntu Linux 12.04.1 LTS
- ✧ メモリ 512MB
- ✧ ProVerif 1.86pl3

5. 備考

本文書は、**総務省「暗号・認証技術等を用いた安全な通信環境推進事業に関する実証実験の請負　成果報告書**」からの引用である。